REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTRE DE L'ENSEIGNEMENT SUPERIEUR ET LA RECHERCHE Scientifique

Université Abou-Bekr Belkaïd – Tlemcen –
Annexe Maghnia
Faculté de Technologie
Département de Sciences et Technologies

Support de cours

Algorithmique

Semestre 1 : ST

2015/2016

Chapitre 1 : Introduction à l'informatique

1. Définitions

Le mot informatique a été proposé par Philippe Dreyfus en 1962 ; c'est un mot-valise, formé d'information et d'automatique. L'informatique c'est donc une automatisation de l'information, plus exactement un traitement automatique de l'information. L'information désigne ici tout ce qui peut être traité par l'ordinateur (textes, nombres, images, sons, vidéos,...).

L'outil utilisé pour traiter l'information de manière automatique s'appelle un ordinateur. Ce nom a été proposé par Jacques Perret (professeur de Latin à La Sorbonne) en 1954. Ce mot était à l'origine un adjectif qui signifiait "qui met de l'ordre", "qui arrange".

L'anglais, plus restrictif, utilise le terme de computer qui peut se traduire par calculateur, machine à calculer.

L'informatique désigne donc un concept, une science, tandis que l'ordinateur est un outil, une machine conçue pour réaliser des opérations informatiques.

2. Familles d'ordinateurs

On distingue généralement plusieurs familles d'ordinateurs selon leur format :

- Les mainframes (en français ordinateurs centraux), ordinateurs possédant une grande puissance de calcul, des capacités d'entrée-sortie gigantesques et un haut niveau de fiabilité.
- Les ordinateurs personnels, parmi lesquels on distingue :

Les ordinateurs de bureau (en anglais desktop computers),

Les ordinateurs portables (en anglais laptop ou notebooks).

- Les tablettes PC (en anglais tablet PC, également appelées ardoises électroniques), composées d'un boîtier intégrant un écran tactile ainsi qu'un certain nombre de périphériques incorporés.
- Les assistants personnels (appelés PDA, pour Personal Digital Assistant, ou encore hand-held, littéralement «tenu dans la main»), parfois encore qualifiés d'organiseur

(en anglais organizer) ou d'agenda électronique, sont des ordinateurs de poche proposant des fonctionnalités liées à l'organisation personnelle.

3. Composants de l'ordinateur

Un ordinateur est un ensemble de composants électroniques modulaires, c'est-à-dire des composants pouvant être remplacés par d'autres composants ayant éventuellement des caractéristiques différentes, capables de faire fonctionner des programmes informatiques. On parle ainsi de « hardware » pour désigner l'ensemble des éléments matériels de l'ordinateur et de « software » pour désigner la partie logicielle.

Un ordinateur se compose essentiellement de :

3.1. Périphériques d'entrée

Le clavier : il permet d'introduire des textes ou encore des données à la machine. On trouve plusieurs types de claviers correspondant à la disposition des touches (lettres, caractères....) afin d'améliorer le confort selon la langue utilisée : on trouve des claviers AZERTY en France et dans beaucoup de pays latins, des claviers QWERTY dans les pays Anglo-saxons, tandis qu'en Allemagne le clavier est de type AWERTY. Dans les pays asiatiques, moyen-orientaux, etc.

On détermine le format du clavier en regardant les six premières touches alphabétiques du clavier décrivant le mot AZERTY ou QWERTY

La souris : Le déplacement de la souris permet de déplacer un curseur sur écran avec lequel (en cliquant sur les boutons) on peut sélectionner, déplacer, manipuler des objets à l'écran.

Le numériseur : Périphérique d'entrée qui permet, par balayage optique, la restitution d'une image à l'écran de l'ordinateur.

3.2. Périphériques de sortie

Le moniteur : Les moniteurs (souvent on utilise le mot écran) se divisent aujourd'hui en deux catégories : les moniteurs à tube cathodique et les moniteurs plats à affichage LCD.

Les caractéristiques : Les moniteurs se caractérisent par les éléments suivants :

• La définition : Le nombre de points affiché. Ce nombre de points est actuellement compris entre 640 × 480 et 1600 × 1200.

• La taille : La taille de l'écran se calcule en mesurant la diagonale de l'écran exprimée en pouces : 1" = un pouce correspond à 2,54 cm.

• La résolution : Elle détermine le nombre de pixels par unité de surface (pixels par pouce carré, en anglais DPI : Dots Per Inch).

• Le pas de masque : C'est la distance qui sépare deux points. Plus celle-ci est petite, plus l'image est précise.

• La fréquence de balayage : C'est le nombre d'images qui sont affichées par seconde. Elle est exprimée en hertz (Hz). Il faut donc qu'elle soit supérieure à 67 hertz (Hz). En dessous de cette limite, l'œil humain remarque que l'image «clignote».

L'imprimante : Il en existe plusieurs types d'imprimantes, dont les plus courantes sont : imprimante laser ; imprimante à jet d'encre ; imprimante matricielle (à aiguilles).

3.3. Unité Centrale

C'est le composant le plus important pour le fonctionnement de l'ordinateur, on peut la trouvé sous différentes forme :

- sous forme de boîtier rectangulaire lequel vient reposer le Moniteur (écran). C'est le desktop

- sous forme de "Moyen tour" qui peut se poser à côté du moniteur

- sous forme de "Grand tour" qui présente l'avantage de pouvoir installer dans des logements prévus en façade, des périphériques supplémentaires comme un graveur de sauvegarde par exemple.

3.3.1. La carte mère

C'est le principal constituant de l'ordinateur. C'est sur cette carte que sont implantés et connectés les autres éléments.

3.3.2. Le microprocesseur

Le premier microprocesseur (Intel 4004) a été inventé en 1971.

Le processeur (CPU) est le cerveau de l'ordinateur, c'est lui qui coordonne le reste des éléments, il se charge des calculs, il exécute les instructions qui ont été programmées. Les principaux éléments d'un microprocesseur sont :

* une horloge qui rythme le processeur. À chaque TOP d'horloge, le processeur effectue une instruction. Ainsi plus l'horloge a une fréquence élevée, plus le processeur effectue d'instructions par seconde (MIPS : Millions d'instruction par seconde). Par exemple un ordinateur ayant une fréquence de 100 mégahertz (MHz) effectue 100 000 000 d'instructions par seconde ;

* une unité de gestion des bus qui gère les flux d'informations entrant et sortant ;

* une unité d'instruction qui lit les données, les décodes puis les envoie à l'unité d'exécution ;

* une unité d'exécution accomplit les tâches données par l'unité d'instruction. La vitesse à laquelle le microprocesseur traite l'information est indiquée en mégahertz ou gigahertz pour les processeurs récents. Deux compagnies se disputent actuellement le marché des processeurs pour les environnements Windows et Linux : AMD (Advanced Micro Devices Inc.) et INTEL.

3.3.3. La mémoire centrale

RAM : La mémoire vive, aussi appelée RAM (Random Access Memory), est contenue dans des barrettes qui se branchent à la carte mère. Elle permet au microprocesseur d'emmagasiner des données pour ensuite y accéder très rapidement. Elle constitue le « bras droit » du microprocesseur et elle est essentielle lors du traitement.

Cette mémoire est cependant temporaire : aussitôt que vous éteignez l'ordinateur, tout son contenu est effacé. Plus il y a de mémoire, plus l'ordinateur est puissant. Un ordinateur qui possède 1 GO de mémoire vive est donc plus puissant et plus rapide qu'un ordinateur qui en possède 512 MO.

Actuellement, la mémoire vive la plus répandue sur le marché des composants informatiques est nommée DDR.

La mémoire cache : La mémoire cache permet au processeur de se «rappeler» les opérations déjà effectuées auparavant. Elle est utilisée par le microprocesseur pour conserver

temporairement des instructions élémentaires. En effet, elle stocke les opérations effectuées par le processeur, afin que celui-ci ne perde pas de temps à recalculer des calculs déjà faits précédemment. La taille de la mémoire cache est généralement de l'ordre de 512 kilo-octets (Ko) voir 1 ou 2 Mo.

La mémoire morte (ROM) : Mémoire permanente contenant des microprogrammes enregistrés sur des puces électroniques de la carte mère (ou mother-board) contenant les routines de démarrage du micro-ordinateur. ROM (Read Only Memory, dont la traduction est mémoire en lecture seule) est appelée aussi parfois mémoire non volatile, car elle ne s'efface pas lors de la mise hors tension du système.

3.3.4. Le disque dur

Le disque dur est l'organe du PC servant à conserver les données de manière permanente, contrairement à la RAM, qui s'efface à chaque redémarrage de l'ordinateur. Il a été inventé au début des années 50 par IBM. C'est également lui qui fournira les informations nécessaires au fonctionnement de la mémoire vive et du processeur lorsque l'ordinateur sera en fonction. Le disque dur moderne peut conserver des quantités phénoménales d'information. On calcule la quantité d'information d'un disque dur en giga-octet (Go), par exemple les disques durs ont des capacités de 40 Go- 240 Go, et actuellement en téraoctets.

Le fonctionnement interne

Un disque dur est constitué de plusieurs disques rigides empilés les uns après les autres à une très faible distance les uns des autres. Ils tournent très rapidement autour d'un axe (à plusieurs milliers de tours par minute actuellement) dans le sens inverse des aiguilles d'une montre. La lecture et l'écriture se font grâce à des têtes (head) situées de part et d'autre de chacun des plateaux (un des disques composant le disque dur). Ces têtes sont des électroaimants qui se baissent et se soulèvent (elles ne sont qu'à quelques microns de la surface, séparées par une couche d'air provoquée par la rotation des disques qui crée un vent d'environ 250 km/h) pour pouvoir lire l'information ou l'écrire. De plus ces têtes peuvent balayer latéralement la surface du disque pour pouvoir accéder à toute la surface...

Le disque dur externe se trouve à l'extérieur du boîtier de l'ordinateur. Les disques durs externes modernes communiquent avec l'ordinateur à l'aide d'une connexion USB.

Les disquettes sont des supports amovibles qui permettent de transporter des documents informatiques peu volumineux, comme un texte en traitement de texte, d'un ordinateur à un autre. Habituellement, une disquette ne peut contenir que 1,44 Mo, ce qui en fait un support très limité pour le transport et le transfert de données.

Le CD-ROM (Compact Disc - Read Only Memory) est un disque optique de 12 cm de diamètre et de 1mm d'épaisseur, permettant de stocker des informations numériques correspondant à 700 Mo

Le DVD-ROM (Digital Versatile Disc - Read Only Memory) est une variante du CD-ROM dont la capacité est largement plus grande, en GO (4,7 – 8).

3.3.5. Les cartes d'extension

La carte son : (en anglais audio card ou soundcard) est l'élément de l'ordinateur permettant de gérer les entrées-sorties sonores de l'ordinateur. Il s'agit généralement d'un contrôleur pouvant s'insérer dans un emplacement ISA ou PCI mais de plus en plus de cartes mères possèdent une carte son intégrée.

La carte graphique : (en anglais graphic adapter), parfois appelée carte vidéo ou accélérateur graphique, est l'élément de l'ordinateur chargé de convertir les données numériques à afficher en données graphiques exploitables par un périphérique d'affichage. .

Carte réseau : La carte réseau est une composante matérielle interne qui permet de connecter votre ordinateur à un réseau informatique. La plupart des ordinateurs neufs possèdent une carte réseau, qui sont identifiées par une seule adresse physique attribuée par IEEE à chaque constructeur.

Modem : Le modem est le périphérique utilisé pour transférer des informations entre plusieurs ordinateurs (2 à la base) via les lignes téléphoniques. Le modem module les informations numériques en ondes analogiques ; en sens inverse il démodule les données numériques. Le modem est l'acronyme de MOdulateur/DEModulateur.

4. Principes de fonctionnement d'un ordinateur

Les deux principaux constituants d'un ordinateur sont la mémoire principale (MC) et le processeur (CPU). La mémoire principale permet de stocker de l'information (programmes et

données), tandis que le processeur exécute pas à pas les instructions composant les programmes.

Notion de programme

Un programme est une suite d'instructions élémentaires, qui vont être exécutées dans l'ordre par le processeur. Ces instructions correspondent à des actions très simples, comme additionner deux nombres, lire ou écrire une case mémoire, etc. Chaque instruction est codifiée en mémoire sur quelques octets. Pour chaque instruction, le processeur effectue schématiquement les opérations suivantes :

1. lire en mémoire (MC) l'instruction à exécuter ;

2. effectuer le traitement correspondant ;

3. passer à l'instruction suivante.

Le processeur est divisé en deux parties, l'unité de commande (UC) et l'unité de traitement :

– l'unité de commande est responsable de la lecture en mémoire et du décodage des instructions.

– l'unité de traitement, aussi appelée Unité Arithmétique et Logique (U.A.L), exécute les instructions qui manipulent les données.

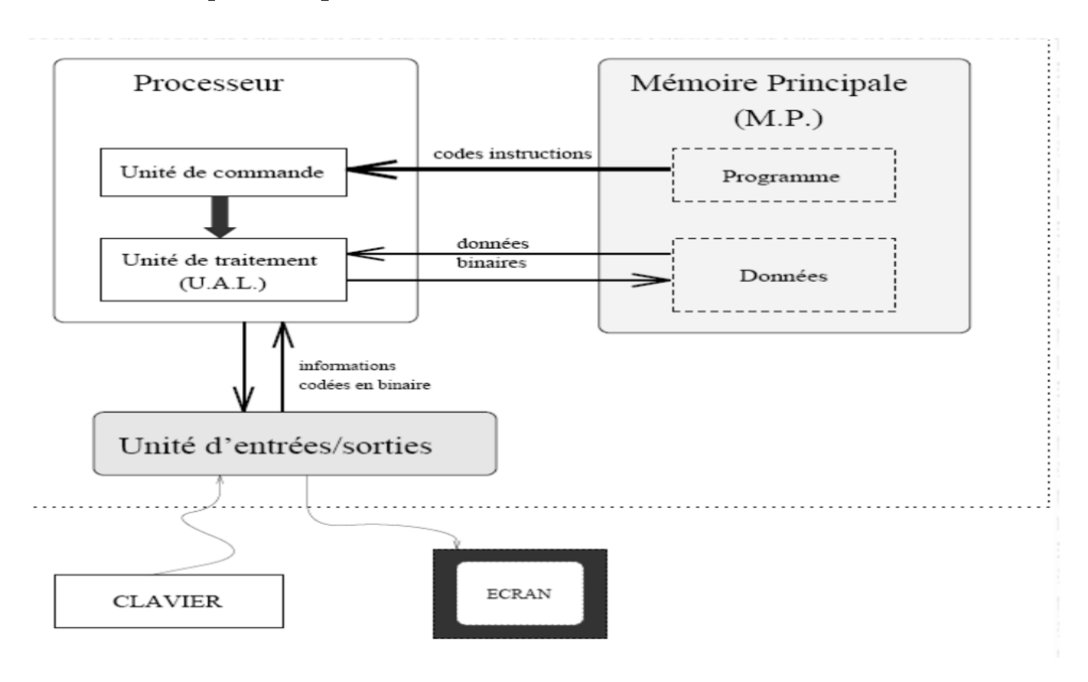


Figure 1 : *Architecture schématique d'un Ordinateur*

5. Les systèmes de codage des informations

Quelle que soit la nature de l'information traitée par un ordinateur (image, son, texte, vidéo), elle est toujours sous la forme d'un ensemble de nombres écrits en base 2, par exemple 01001011.

Le terme bit (b minuscule dans les notations) signifie « binary digit », c'est-à-dire 0 ou 1 en numérotation binaire. Il s'agit de la plus petite unité d'information manipulable par une machine numérique. Il est possible de représenter physiquement cette information binaire par un signal électrique ou magnétique, qui, au-delà d'un certain seuil, correspond à la valeur 1.

L'octet (en anglais byte ou B majuscule dans les notations) est une unité d'information composée de 8 bits. Il permet par exemple de stocker un caractère comme une lettre ou un chiffre.

Une unité d'information composée de 16 bits est généralement appelée mot (en anglais word). Une unité d'information de 32 bits de longueur est appelée mot double (en anglais double word, d'où l'appellation dword). Voici les unités standardisées :

- Un kilooctet (ko) = 1000 octets
- Un mégaoctet (Mo) = 10^6 octets
- Un gigaoctet (Go) = 10^9 octets
- Un téraoctet (To) = 10^{12} octets
- Un pétaoctet (Po) = 10^{15} octets

5.1. Les systèmes de numération

5.1.1 La conversion d’un nombre de la base 10 vers une base b

Pour convertir un nombre (N) en décimal vers une autre base (b) il suffit d’appliquer l’algorithme de division successive du nombre N par b et de prendre les restes dans le sens inverse (le dernier reste est le poids fors, c'est-à-dire le plus à gauche, et le premier reste est le poids faible, c'est-à-dire le plus à droite).

5.1.2. La conversion d'un nombre d'une base b vers la base 10

Pour convertir un nombre M = $(a_{n-1}\ a_{n-2}\ \ldots\ a_2\ a_1\ a_0)_b$, qui comporte n termes (chiffres) écrit dans une base b, il suffit d'additionner les termes a_i multiplier par la base b à la puissance i qui représente le rang du terme.

N = $a_{n-1} \times b^{n-1} + a_{n-2} \times b^{n-2} + \ldots + a_2 \times b^2 + a_1 \times b^1 + a_0 \times b^0$

N= $\sum_{i=0}^{n-1} a_i \times b^i$

i : est le rang du terme dans le nombre (de droite (0) vers la gauche (n-1))

b : la base du nombre M

a_i : sont les termes du nombre M

5.1.3. Les bases décimale, binaire, octale et hexadécimale

Le tableau ci-dessous montre la représentation des nombres de 0 à 15 dans les bases 10, 2 et 16.

Décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binaire	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Hexadécimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

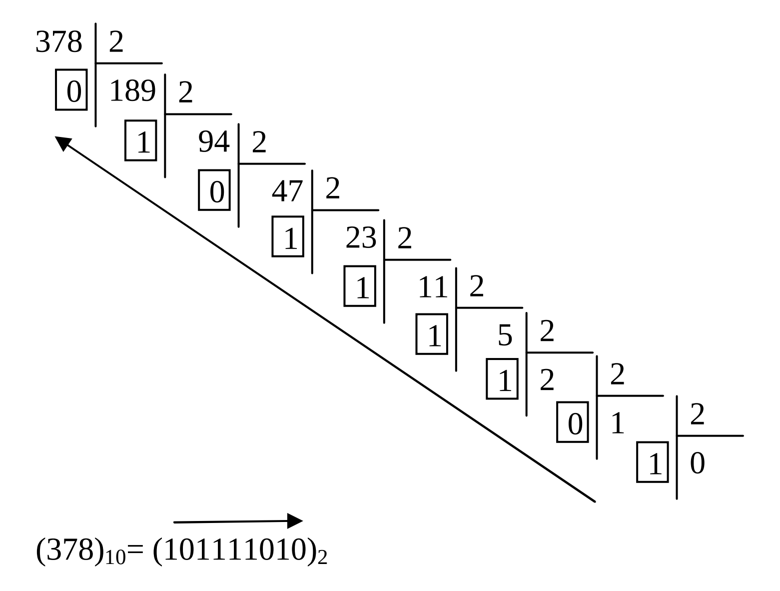
5.1.3.1. Conversion décimal – binaire

Elle est obtenue par divisions successives par 2 et on prend les restes dans le sens inverse.

$(378)_{10}$ = $(?)_2$
378 : 2 = 189 reste 0 ↑ Poids faible (le plus à droite)
189 : 2 = 94 reste 1
94 : 2 = 47 reste 0
47: 2 = 23 reste 1
23 : 2 = 11 reste 1
11 : 2 = 5 reste 1
5 : 2 = 2 reste 1
2 : 2 = 1 reste 0
1 : 2 = 0 reste 1 Poids fort (le plus à gauche)

→

$(378)_{10}$= $(101111010)_2$



$(378)_{10}= (101111010)_2$

5.1.3.2. Conversion binaire- décimal

$(100110111001)_2=(?)_{10}$

	11	10	9	8	7	6	5	4	3	2	1	0	les rangs *i*
$(100110111001)_2$=(	1	0	0	1	1	0	1	1	1	0	0	1)$_2$	la base =2
	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	les termes (chiffres) a_i

$(100110111001)_2=(1 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)_{10}$

$= (2048 + 0 + 0 + 256 + 128 + 0 + 32 + 16 + 8 + 0 + 0 + 1)_{10}$

$= (2489)_{10}$

5.1.3.3. Conversion hexadécimal/binaire

Pour convertir un nombre binaire en hexadécimal, il suffit de faire des groupes de quatre (04) bits (en commençant par la droite).

$(1001101)_2 = (\underline{0100}\ \underline{1101})_2 = (4D)_{16}$

Tableau 1 : Correspondance Binaire/Hexadécimal

Binaire	Hexadécimal
0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**

Binaire	Hexadécimal
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**

Et pour la conversion d'un nombre hexadécimal en binaire, il suffit de convertir chaque chiffre hexadécimal en binaire codé sur quatre (04) bits :

$(13B0F5)_{16} = (0001\ 0011\ 1011\ 0000\ 1111\ 0101)_2 = (10011101100000111110101)_2$

5.1.3.4. Conversion octal/binaire

Pour convertir un nombre binaire en octal, il suffit de faire des groupes de trois (03) bits (en commençant par la droite).

$(1001101)_2 = (\underline{001}\ \underline{001}\ \underline{101})_2 = (115)_8$

Tableau 2 : Correspondance Binaire/Octal

Binaire	**Octal**
000	**0**
001	**1**
010	**2**
011	**3**
100	**4**
101	**5**
110	**6**
111	**7**

Et pour la conversion d'un nombre octal en binaire, il suffit de convertir chaque chiffre octal en binaire codé sur trois (03) bits :

$(13705)_8 = (001\ 011\ 111\ 000\ 101)_2 = (1011111000101)_2$

5.2. Représentation de l'information

5.2.1. Représentation des nombres entiers

5.2.1.1. Représentation d'un entier naturel

Un entier naturel est un nombre entier positif ou nul, pour le représenter il suffit de le convertir en binaire sur le nombre de bits du code.

Sur 8 bits on peut coder de [0 , 255]

Sur n bits on peut coder $[0 , 2^n-1]$

5.2.1.2. Représentation d'un entier relatif

Code binaire signe (par signe et valeur absolue)

Pour coder des nombres relatifs sur n bits on utilise le bit le plus significatif pour représenter le signe du nombre :

- si le bit le plus fort = 1 alors nombre négatif
- si le bit le plus fort = 0 alors nombre positif

Et les autres (n-1) bits pour coder la valeur absolue du nombre.

Sur 8 bits on peut coder de [-127 , +127]

Sur n bits on peut représenter : $[-(2^{n-1}-1), +(2^{n-1}-1)]$

Exemple :

Représentation de (-14) sur 8 bits en signe + valeur absolue.

$14 = (1110)_2 = (0000\ 1110)_2 \rightarrow$ S+VA(-14) : 1 000 1110
Sur 8 bits
S VA

Signe → S ; Valeur Absolue → VA

Code complément à 1 (Complément Logique)

- Les nombres positifs sont codes de la même façon qu'en binaire pure.
- Un nombre négatif est codé en inversant chaque bit de la représentation de sa valeur absolue

Le bit le plus significatif est utilise pour représenter le signe du nombre :

- Si le bit le plus fort = 1 alors nombre négatif
- Si le bit le plus fort = 0 alors nombre positif

Sur 8 bits on peut coder de [-127 , +127]

Sur n bits on peut représenter : $[-(2^{n-1}-1), +(2^{n-1}-1)]$

Exemple :

Représentation de (-14) sur 8 bits en Complément à 1.

14 = 0 0 0 0 1 1 1 0
Cà1(-14) = 1 1 1 1 0 0 0 1 inverser les bits

Code complément à 2 ou Complément arithmétique (Utilise sur ordinateur)

- les nombres positifs sont codes de la même manière qu'en binaire pure.

- un nombre négatif est code en ajoutant la valeur 1 à son complément à 1
- Le bit le plus significatif est utilise pour représenter le signe du nombre

Sur 8 bits on peut coder de [-128 , +127]

Sur n bits on peut représenter : [-2^{n-1}, $+(2^{n-1}-1)$]

Exemple :

Représentation de (-14) sur 8 bits en Complément à 2.

14 = 0 0 0 0 1 1 1 0
Cà1(-14) = 1 1 1 1 0 0 0 1 inverser les bits
Cà2(-14) = 1 1 1 1 0 0 1 0 Cà1(-14) +1

5.2.1.3. Représentation d'un nombre réels

Format virgule fixe

Utilisée par les premières machines. Possède une partie ‘entière’ et une partie ‘décimale’ séparés par une virgule. La position de la virgule est fixe d’ou le nom.

Exemple :

$(54,25)_{10} = (\ ?\ )_2$

Partie entière : méthode de division

$(54)_{10} = (110110)_2$

Partie décimale : méthode de multiplication

0,25
x 2
= 0, 50
0,50
x 2
= 1, 00

$(0,25)_{10} = (0,01)_2$

$(54,25)_{10} = (110110,01)_2$

Conversion vers la base 10 :

Etant donne une base b

- un nombre x est représenté par :

$x = (a_{n-1}\ a_{n-2}\ldots a_1\ a_0,a_{-1}\ a_{-2}\ldots a_{-p})_b$
a_{n-1} : est le chiffre de poids fort
a_{-p} : est le chiffre de poids faible
n : est le nombre de chiffre avant la virgule
p : est le nombre de chiffre après la virgule

La valeur de x en base 10 est : $x = \sum_{i=-p}^{n-1} a_i \times b^i$

Exemple :

$(10011011,1001)_2=(\ ?\ )_{10}$

	7	6	5	4	3	2	1	0		-1	-2	-3	-4	les rangs i
$(10011011,1001)_2=($	1	0	0	1	1	0	1	1	,	1	0	0	$1\ )_2$	la base =2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0		a_{-1}	a_{-2}	a_{-3}	a_{-4}	les termes (chiffres) a_i

$(10011011,1001)_2=(\ 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}\ )_{10}$

$= (\ 128 + 0 + 0 + 16 + 8 + 0 + 2 + 1 + 0{,}50 + 0 + 0 + 0{,}0625\ )_{10}$

$= (155{,}5625)_{10}$

Format virgule flottante

Utilisée actuellement sur machine

défini par : $\pm\ m\ .\ b^{\ e}$

- un signe + ou –
- une mantisse m (en virgule fixe)
- un exposant e (un entier relative)
- une base b (2, 8, 10, 16,…)

Le codage en base 2, format virgule flottante, revient a coder le signe, la mantisse et l'exposant.

$x = \pm\ m\ .\ 2^{\ e}$

La normalisation :

$X = \pm\ 1{,}M\ .\ 2^{\ E}$

1- Le signe est code sur 1 bit ayant le poids fort :

- Le signe – : bit 1

- Le signe + : bit 0

2- Exposant biaise (E)

- placé avant la mantisse pour simplifier la comparaison
- Code sur p bits et biaise pour être positif (ajout de 2^{p-1}-1)

3- écrire la mantisse m sous forme normalisée (m=1,M)

- Normalise : virgule est place après le bit à 1 ayant le poids fort
- M est code sur q bits

Exemple : 11,01→ m= 11,01 = 1,101 = 1,M donc M =101

S	Exposant E	Mantisse M
1bit	p bits	q bits

Standard IEEE 754

Simple précision sur 32 bits :

S	**E**	**M**
1bit	8 bits	23 bits

- 1 bit de signe de la mantisse
- 8 bits pour l'exposant (E = e +127)
- 23 bits pour la mantisse

Double précision sur 64 bits :

S	**E**	**M**
1bit	11 bits	52 bits

- 1 bit de signe de la mantisse
- 11 bits pour l'exposant (E = e + 1023)
- 52 bits pour la mantisse

Exemple :

35,5(10) = ?$_{\text{(IEEE 754 simple précision)}}$

Nombre positif, donc S = 0

35,5(10) = $(100011,1)_2$ (virgule fixe)

= $(1,000111 \,.\, 2^5)_2$ (virgule flottante) exposant e=5

Exposant E = e +127 = 5 + 127 = 132_{10} = $(1000\ 0100)_2$

Mantisse m = 1,000111 = 1,M = 1,000111 donc M = 00011100...

La norme IEEE en simple précision (IEEE 754 SP)

0	**1000 0100**	**000 1110 0000 0000 0000 0000**
1bit	8 bits	23 bits

5.2.1.4. Représentation des caractères

Le codage revient à créer une Table de correspondance entre les caractères et des nombres.

- **Code ASCII (American Standard Code for Information Interchange)**

est un code sur 7 bits pour représenter 128 caractères (0 à 127)

de 48 à 57 : chiffres dans l'ordre (0,1,...,9)

de 65 à 90 : les alphabets majuscules (A,...,Z)

de 97 à 122 : les alphabets minuscule (a,...z)

- **Le code ASCII Etendu**

8 bits pour représenter 256 caractères (0 à 255)

Code les caractères accentués : à, è,...etc.

- **Code Unicode :**

Mis au point en 1991 (codage sur internet),

16 bits pour représenter 65 536 caractères (0 à 65 535)

Code la plupart des alphabets : Arabe, Chinois, ...

On en a défini environ 50 000 caractères pour l'instant

6. Le SOFTWARE (Partie Logiciel d'un Ordinateur) :

Le software se décompose en trois principales familles :

- **logiciels d'application:**

Résolution de problèmes spécifiques (traitement de textes, PAO, tableurs, logiciels de comptabilité, CAO (conception, design et simulation), ...).

- **logiciels utilitaires:**

Logiciels qui servent au développement des applications (assembleur, compilateurs, gestionnaire de fenêtres, librairie de dessin, outils de communications...).

- **logiciels systèmes (le système d'exploitation):**

Présents au cœur de l'ordinateur, ces logiciels sont à la base de toute exploitation,

coordonnant les tâches essentielles à la bonne marche du matériel.

C’est du système d’exploitation que dépend la qualité de la gestion des ressources (processeur, mémoire, périphériques) et la convivialité de l’utilisation d’un ordinateur.

6.1. Définition d’un système d’exploitation

Le système d’exploitation est l’ensemble des programmes qui se chargent de résoudre les problèmes relatifs à l’exploitation de l’ordinateur. Ils tournent en permanence sur un ordinateur et le contrôlent à partir de son démarrage (boot), son rôle est de réaliser deux tâches distinctes :

• Gérer les ressources physiques de l’ordinateur

Assurer l’exploitation efficace, fiable et économique des ressources critiques (processeur, mémoire…)

• Gérer l’interaction avec les utilisateurs

Faciliter le travail des utilisateurs en leur présentant une machine plus simple à exploiter que la machine réelle.

6.2. Classification des systèmes d’exploitation:

- **Mono-tâche**

A tout instant, un seul programme est exécuté; un autre programme ne démarrera, sauf conditions exceptionnelles, que lorsque le premier sera terminé. Exemple: DOS

- **Multi-tâches**

Plusieurs processus (i.e. un «programme» en cours d’exécution) peuvent s’exécuter simultanément (systèmes multi-processeurs) ou en quasi-parallélisme (systèmes à temps partagé). Exemple: Windows.

- **Mono-session**

Au plus un utilisateur à la fois sur une machine. Les systèmes réseaux permettent de différencier plusieurs utilisateurs, mais chacun d’eux utilise de manière exclusive la machine (multi-utilisateurs, mono-session). Exemple: Windows

- **multi-sessions**

Plusieurs utilisateurs peuvent travailler simultanément sur la même machine. Exemple: VMS, Unix

6.3. Exemple de Système d'Exploitation :

- Unix : Créé en 1969, rapidement multi-utilisateur, écrit en langage C.
- Linux : Clone gratuit d’UNIX pour les PC, open source.
- Mac OS : Premier à proposer le concept des fenêtres, du glisser-déposer, la corbeille, le plug-and-play; aujourd’hui possède le noyau Linux, avec une interface graphique élégante et ergonomique, et optimisation particulière des traitement multimédia.
- MS-DOS (Microsoft disque operating system) : SE des premiers PC, mono-utilisateur, mono-tâche, interface ligne de commande.
- MS-Windows : Inspiré par l’interface Macintosh; tout d’abord, une coquille graphique pour DOS. Seulement à partir de Windows 95 nous commençons à assister à un transfert de nombreuses fonctionnalités de DOS vers Windows.

6.4. Structure en couche d’un SE :

- **Le noyau**

Les fonctions principales du noyau sont:

- Gestion du processeur ;
- Gestion des interruptions ;
- Gestion du multi-tâches,

- **Système de Fichiers**

Le concept de fichiers est une structure adaptée aux mémoires secondaires et auxiliaires permettant de regrouper des données.

Le rôle d’un système d’exploitation est de matérialiser le concept de fichiers (le gérer, c’est-à-dire le créer, le détruire, l’écrire (modifier) et le lire.

Dans le cas de systèmes multi-utilisateurs, il faut de plus assurer la confidentialité de ces fichiers, en protégeant leur contenu du regard des autres utilisateurs.

- **Entrées-Sorties**

Il s’agit de permettre le dialogue (échange d’informations) avec l’extérieur du système.

Concrètement, la gestion des E/S implique que le SE mette à disposition de l’utilisateur des procédures standard pour l’émission et la réception des données, et qu’il offre des traitements appropriés aux multiples conditions d’erreurs susceptibles de se produire (plus de papier, erreur de disque, débit trop différent, ...).

6.5. Interpréteur de commandes

Pour interagir avec l’utilisateur, un système informatique doit disposer d’un interpréteur de commandes (shell).

Le shell attend les ordres que l’utilisateur transmet par le biais de l’interface, décode et décompose ces ordres en actions élémentaires, et finalement réalise ces actions en utilisant les services des couches plus profondes du système d’exploitation.

Parmis les shells Unix les plus utilisés, citons:

Bourne [Again] shell (sh et bash), C shell (csh), Korn shell (ksh), Z shell (zsh), et celui présent par défaut sur les comptes du cours, l’Enhanced C shell (tcsh).

Chapitre 2: Notions d'Algorithme et de Programme

1. Concept d'un algorithme

1.1. Définition

L’Encyclopédie Universalis donne la définition suivante de l’Algorithme :

« Un algorithme est une suite de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver, en un nombre fini d’étapes, à un certain résultat, et cela indépendamment des données. »

– Le mot « Algorithme » est inventé par le mathématicien « ALKHAWARISMI ». Un Algorithme est l'énoncé d'une séquence d'actions primitives réalisant un traitement. Il décrit le plan ou les séquences d'actions de résolution d'un problème donné.

– Un algorithme est un ensemble d'actions (ou d'instructions) séquentielles et logiquement ordonnées, permettant de transformer des données en entrée (Inputs) en données de sorties (outputs ou les résultats), afin de résoudre un problème.

Donc, un algorithme représente une solution pour un problème donné. Cette solution est spécifiée à travers un ensemble d'instructions (séquentielles avec un ordre logique) qui manipulent des données. Une fois l'algorithme est écrit (avec n'importe quelle langues : français, anglais, arabe, etc.), il sera transformé, après avoir choisi un langage de programmation, en un programme code source qui sera compilé (traduit) et exécuté par l'ordinateur.

1.2. Propriétés d'un algorithme

- L’algorithme doit tenir compte de tous les cas possibles ;
- Il contient toujours un nombre fini d'actions ;
- L’ordre des actions est important (exécution séquentielle) ;
- Chaque action doit être définie avec précision, sans aucune difficulté ;
- L’algorithme n’est pas nécessairement unique ;
- Il doit produire le résultat désiré.

1.3. Exemple :

Algorithme 1 : Ouvrir la fenêtre

1- Aller vers la fenêtre

2- tourner le poignet

3- tirer la fenêtre.

Algorithme 2 : préparer frittes

1. Laver 4 pommes de terre.
2. Éplucher les pommes de terre.
3. Les couper en tranches de 1 cm d'épaisseur
4. puis chaque tranche en bâtonnets carrés.
5. faire chauffer de l'huile dans une poile ou dans la friteuse
6. cuire les frites pendant 5 à 8 min dans l'huile

Algorithme 3 : somme de deux nombre

1. Saisir le premier nombre
2. Saisir le deuxième nombre
3. Faire la somme
4. Restituer le résultat

1.4. Structure d'un algorithme

Un algorithme est composé de trois parties :

L'entête : comporte le mot 'ALGORITHME' et le nom de l'algorithme

Environnement : déclaration des variables utilisées dans cet algorithme, comporte le mot **CONSTANTE**, **VARIABLE** suivi de la liste des variables et leurs types.

Le corps de l'algorithme : regroupe l'ensemble des instructions et il est délimité par les mots **DEBUT** et **FIN**.

Exemple :

```
ALGORITHME somme
VARIABLE
        a, b, c : ENTIER
DEBUT
        ECRIRE ('donner deux valeurs')
        LIRE(A)
        LIRE(B)
        C ←— A+B
        ECRIRE(C)
FIN.
```

2. Représentation en organigramme

Un organigramme est un schéma symbolique conventionnel qui illustre les étapes d'un algorithme et leurs relations.

Nous utilisons l'organigramme parce qu'une représentation graphique aide à la compréhension.

L'organigramme est un schéma fonctionnel qui présente les différentes parties d'un programme les unes à la suite des autres en utilisant des symboles graphiques pour visualiser l'exécution du programme et le cheminement des données.

Tableau 3 : Principaux symboles d'un organigramme

Nom	Symbole	Définition
Flèches		Elles indiquent le sens du traitement (haut, bas, gauche, droite).
Début / Fin		Ce symbole indique le début ou la fin de l'organigramme
Entrée / Sortie		Ce symbole indique les données d'entrées et de sorties
Boite de Traitement		Elle indique un traitement spécifique qui peut être exécuté
Boite de décision (test)		Elle permet d'envoyer le traitement sur un chemin ou sur un autre, selon le résultat du test

3. Structure d'un programme

3.1. Définition

Programme

Un programme informatique est le codage lisible par l'ordinateur d'un algorithme. C'est une succession d'instructions exécutables par l'ordinateur.

Langage de programmation

Le langage de programmation est l'intermédiaire entre l'humain et la machine, il permet d'écrire dans un langage proche de la machine mais intelligible par l'humain les opérations que l'ordinateur doit effectuer.

Étant donné que le langage de programmation est destiné à l'ordinateur, il doit donc respecter une syntaxe stricte.

Compilation

La transformation d'un programme en langage machine est appelé la compilation.
La compilation est une phase réalisée par l'ordinateur lui-même grâce à un autre programme appelé compilateur.



3.2. Les différents Langages (du plus haut niveau vers bas niveau)

- Python, Rebol, Ruby
- Php, Perl, Tcl
- C++, Java
- Pascal
- C, Fortran
- Assembleur
- Langage machine

4. La démarche et analyse d'un problème

Pour atteindre à cette solution algorithmique un processus d'analyse et de résolution sera appliqué. Ce processus est constitué des étapes suivantes :

- Analyser ce problème : définir avec précision les résultats à obtenir, les informations dont on dispose, ...
- Déterminer les méthodes de résolution : il s'agit de déterminer la suite des opérations à effectuer pour obtenir à partir des données la solution au problème posé. Cette suite d'opérations constitue un algorithme.

- Formuler l'algorithme définitif : cette étape doit faciliter la résolution sur ordinateur par l'expression de l'algorithme dans un formalisme adéquat.
- Traduire l'algorithme dans un langage de programmation adapté.

PROBLÈME : Analyse / étude du problème à résoudre.

MODÈLE : Spécifier le modèle de résolution

ALGORITHME : Écriture de l'algorithme

PROGRAMME : Traduction de l'algorithme en programme

RÉSULTATS : Exécution du programme par l'ordinateur.

5. Structure des données

Un algorithme permet de réaliser un traitement sur un ensemble de données en entrées pour produire des données en sorties. Les données en sorties représentent la solution du problème traité par l'algorithme.

Toutes les données d'un programme sont des objets dans la mémoire vive (c'est un espace réservé dans la RAM). Chaque objet (espace mémoire) est désigné par une appellation dite : ***identificateur***.

5.1- Notion d'identificateur

Un identificateur est une chaîne de caractères contenant uniquement des caractères alphanumériques (alphabétiques de [a..z] et [A..Z] et numérique [0..9]) et tiré 8 '_' (blanc souligné), et qui doit commencer soit par un une lettre alphabétique ou _.

Un identificateur permet d'identifier d'une manière unique un algorithme (ou un programme), une variable, une constante, une procédure ou une fonction.

Dans un langage de programmation donnée, on n’a pas le droit d'utiliser les mots réservés (mots clés) du langage comme des identificateurs.

Mots réservés du langage Pascal

```
AND, ARRAY, ASM, BEGIN, CASE, CONST, CONSTRUCTOR, DESTRUCTOR,
DIV, DO, DOWNTO, ELSE, END, EXPORTS, FILE, FOR, FUNCTION,
GOTO, IF, IMPLEMENTATION, IN, INHERITED, INLINE, INTERFACE,
LABEL, LIBRARY, MOD, NIL, NOT, OBJECT, OF, OR, PACKED,
PROCEDURE, PROGRAM, RECORD, REPEAT, SET, SHL, SHR, STRING,
THEN, TO, TYPE, UNIT, UNTIL, USES, VAR, WHILE, WITH, XOR.
```

Exemples :

a1 : est un identificateur valide.

a_1 : est un identificateur valide.

A_1 : est un identificateur valide.

x12y : est un identificateur valide.

x1 y : est un identificateur non valide (à cause du blanc ou l'espace).

x1-y : est un identificateur non valide (à cause du signe -).

x1_y : est un identificateur valide.

1xy : est un identificateur non valide (commence un caractère numérique).

5.2. Constantes et variables

Les données manipulées par un algorithme (ou un programme) sont soit des constantes ou des variables :

– Constantes : une constante est un objet contenant une valeur qui ne peut jamais être modifiée. Son objectif est d'éviter d'utiliser une valeur d'une manière directe. Imaginons qu'un algorithme utilise la valeur 3.14 une dizaine de fois (le nombre d'occurrences de la valeur 3.14 est par exemple 15) et qu'on veut modifier cette valeur par une autre valeur plus précise : 3.14159. Dans ce cas on est amené à modifier toutes les occurrences de 3.14. Par contre, si on utilise une constante PI = 3.14 on modifier une seule fois cette constante.

– Variables : une variable est un objet contenant une valeur pouvant être modifiée. Toutes les données (variable ou constante) d'un algorithme possèdent un type de données (domaine de valeurs possibles).

5.3. Types de données

Dans l'algorithmique, nous avons cinq types de base :

– Entiers : représente l'ensemble {…, -4, -3, -2, -1, 0, 1, 2, 3, 4, ...}

– Réels : représente les valeurs numériques fractionnels et avec des virgule fixes (ou flottante)

– Caractères : représente touts les caractères imprimable.

– Chaînes de caractères : une séquence d'un ou plusieurs caractères

– Booléens (logique) : représente les deux valeurs (VRAI) TRUE et (FAUX) FALSE.

5.4. Traduction d’un algorithme en programme pascal

Algorithme <identificateur_algo> <Declarations> **Début** <Corps> **Fin**.	**Program** prog1; <declarations>; **Begin** <Instructions>; **End.**

- Un programme commence par le mot clé « **program** » suivi d’un identificateur qui représente le nom du programme ;
- Le bloc déclaration englobe les différentes forme de la déclaration (label, constante, variable…) ;
- Le corps du programme commence par le mot clé (réservé) « **begin** » et se termine par le mot clé « **end.** » suivi d’un point et englobe l’ensemble des instructions du programme ;
- Chaque instruction en pascal se termine par un point virgule « ; »

Le tableau suivant montre la correspondance entre les types d'algorithme et les types du langage de programmation PASCAL.

Tableau 4 : Correspondance des types d'algorithme et les types en PASCAL

Algorithme	***PASCAL***
Entier	Integer
Réel	Real
Booléen	Boolean
Caractère	Char
chaîne	String

Tableau 5 : Le type entier en pascal

Type	Intervalle	Taille	
Byte	[0 .. 255]	1 octet	Nombre Non signé
Word	[0 .. 65535]	2 octets	
LongWord	[0 .. 4294967295]	4 octets	
QWord	[0 .. 18446744073709551615]	8 octets	
Shortint	[-128 .. +127]	1 octets	Nombre signé
Smallint	[-32768 .. 32767]	2 octets	
Integer	Smallint ou longint	2/4 octets	
Longint	[-2147483648 .. +2147483647]	4 octets	
Int64	[-9223372036854775808 .. +9223372036854775807]	8 octets	

Tableau 6 : Le type réel en pascal

Type	Intervalle	Précision	Taille
Real	1.5E-45 .. 3.4E38	7-8	4 octets
Single	1.5E-45 .. 3.4E38	7-8	4 octets
Double	5.0E-324 .. 1.7E308	15-16	8 octets
Extended	1.9E-4932 .. 1.1E4932	19-20	10 octets

6. Les opérateurs

6.1. L'opérateur d'affectation

L'affectation est instruction (se termine par ; en pascal) qui permet d'attribuer une valeur à une variable.

- La variable doit ce trouvé à gauche de l'opérateur affectation ;
- la valeur se trouve à droite de l'opérateur et peut être (une constante, le contenu d'une variable ou l'évaluation d'une expression).

Syntaxe :

Variable ⟵ valeur | constante | variable | expression

En algorithmique on utilise le symbole : ⟵ qui indique le sens d'attribution

En pascal on utilise le symbole **:=**

L'expression doit être une expression bien formée.

Exemple :

Algorithmique	Pascal
A ← 5	A := 5 ;
B ← 5+2	B :=5+2 ;
C ← B	C := B ;
D ← A+5*2+B/C	D := A + 5 * 2 + B / C ;

Les expressions :

Une expression désigne une valeur, exprimée par composition d'opérateurs appliqués à des opérandes, qui sont : des valeurs, des constantes, des variables, des appels de fonction ou des sous-expressions.

Syntaxe :

Certains opérateurs agissent sur 2 opérandes : operande1 **operateur_binaire** operande2

D'autres agissent sur 1 opérande : **operateur_unaire** operande

- Les opérateurs binaires sont :
  - – opérateurs de relation : =, <>, <=, <, >, >=
  - – opérateurs additifs : +, -, or
  - – opérateurs multiplicatifs : *, /, div, mod, and
- Les opérateurs unaires sont :
  - – opérateurs de signe : +, -
  - – opérateur de négation not
- Les parenthèses sont un opérateur primaire, elles peuvent encadrer tout opérande.
- Une fonction est aussi un opérateur primaire, elle agit sur l'opérande placé entre parenthèses à sa droite. Certaines fonctions ont plusieurs paramètres, séparés par des virgules.

6.2. Les opérateurs arithmétiques

Opérateur	**Signification**
+	Addition – Union
-	Soustraction – Complément
*	Multiplication – Intersection
/	Division
div	Quotient de la division entière
mod	Modulos : c'est le reste de la division entière

6.3. Les opérateurs relationnels

Opérateur	Signification
<	Inférieur strict
<=	Inférieur ou égal – Inclus
>	Supérieur strict
>=	Supérieur ou égal – Contient
<>	Différent
=	Egalité

6.4. Les opérateurs logiques

Opérateur	Signification	En pascal
Et	Le "et" logique des maths	AND
Ou	Le "ou"	OR
Ou exclusif	Le "ou" exclusif	XOR
Non	Le "non"	NOT

6.5. Les priorités dans les opérations

Priorité des opérateurs

Ordre de priorité	Opérateur	Type
Niveau 1	() , fonction()	Primaire
Niveau 2	+, - , NOT	Unaire
Niveau 3	*, /, mod, div, AND	Multiplicatif
Niveau 4	+, -, OR, XOR	Additif
Niveau 5	=, <, >, <=, >=, <>	Relation

Une expression (arithmétique ou logique) s’évalue suivant les règles :

1- Toute expression s’évalue de gauche à droite.
2- Evaluer en premier l’opération dont l’opérateur est prioritaire.
3- Les parenthèses augmentent la priorité d’un opérateur.

<u>Exemple :</u>

1- A := 1+6*3-4/2

5+6*3-4/2 (priorité pour * et / par rapport au + et -, commencer par * il est plus à gauche. Donc on calcule 6*3=18)

= 1+18-4/2 (priorité pour / par rapport au + et -. Donc on calcule 4/2=2)

= 1+18-2 (+ et – ont même priorité, commencer par l’opérateur le plus à gauche. Donc on calcule 1+18=19)

=19-2=17 (A=17).

2- B := 1+6*(3-4)/2

1+6*(3-4)/2 = 1+6* -1/2 (- set l’opérateur unaire qui nous donne 6*(-1)=-6

= 1+ -6 /2 = 1+ -3= -2 (B=-2)

3- C :=3<5 or 6=2 (cette expression est mal formée, sans les parenthèses elle donne erreur. L’opérateur OR est prioritaire alors on évalue 5 OR 6 erreur car le OR s’applique sur des opérandes logiques et non entiers)

L’expression bien formée est : (3<5) or (6=2)

(3<5) or (6=2) = Vrai or (6=2) = Vrai or Faux =Vrai

6.6. Les fonctions standards

Fonctions mathématiques Pascal de base :

Fonction	Signification	Exemple
ABS	renvoie la valeur absolue	ABS(-5)=5
SQR	renvoie le carré	SQR(4)=16
SQRT	racine carrée	SQRT(64)=8
EXP	Exponentielle	
LN	log népérien	
SIN	Sinus	
COS	Cosinus	
ARCTAN	arc tangente	
ROUND	arrondi à l'entier le plus proche	ROUND(15.9)=16
TRUNC	partie entière (permet de mettre un réel dans un entier)	TRUNC(4.5) = 4
FRAC	Prise de la partie fractionnaire du nombre réel	Frac(4.531)=0.531
INT	Prise de la partie entière du nombre sans arrondis	INT(4.5)=4.00

Autres fonctions :

Fonction	Signification
Inc(a)	Le nombre a est incrémenté de 1
Inc(a,n)	Le nombre a est incrémenté de n
Dec(a)	Le nombre a est décrémenté de 1
Dec(a,n)	Le nombre a est décrémenté de n
SUCC(a)	renvoie le successeur de la variable passée en paramètre
PRED(a)	renvoie le prédécesseur de la variable passée en paramètre
ORD(a)	renvoie l'index de la variable dans l'intervalle auquel elle appartient
High(a)	Renvoie la plus grande valeur possible que peut prendre de la variable a
Low(a)	Renvoie la plus petite valeur possible que peut prendre de la variable a
Odd(a)	Renvoie true si le nombre a est impair et false si a est pair
SizeOf(a)	Renvoie le nombre d'octets occupés par la variable a

Exemple :

```
program exemple;
var
   entier, a, b, c, d, nb_octet, max_int, min_int : integer;
   reel :real;
   Long, max_long, min_long:longint;
   logique : boolean;
begin
   entier:=ord('c');
     Writeln('le code ASCII de "c" est ',entier);
   b:=5;
   entier:=ord(b);
     Writeln('l''ordre de b dans l''ensemble des entiers est ',entier);
   entier:=abs(-5);
     Writeln('la valeur absolue de -5 = ',entier);
   reel:=abs(-5.4);
     Writeln('la valeur absolue de -5.4 = ',reel:0:2);
   entier:=sqr(4);
     Writeln('le carre de 4 = ',entier);
   reel:=sqr(4.2);
     Writeln('le carre de 4.2 = ',reel:0:2);
   reel:=sqrt(4);
     Writeln('la racine carre de 4 = ',reel:0:2);
   reel:=sqrt(4.2);
     Writeln('la racine carre de 4.2 = ',reel:0:2);
   entier:=round(15.9);
     Writeln('l''arrondi de 15.9 = ',entier);
   entier:=round(15.5);
     Writeln('l''arrondi de 15.5 = ',entier);
   entier:=round(15.2);
     Writeln('l''arrondi de 15.2 = ',entier);
   long:=trunc(18954235.758);
     Writeln('partie entiere de 18954235.758 = ',long);
   reel:=frac(14.531);
     Writeln('partie entiere de 14.531 = ',reel:0:5);
   reel:=int(105.92);
     Writeln('partie entiere de 105.92 = ',reel:0:2);
   a:=9;  b:=9;  c:=9;  d:=9;  entier:=9;
   inc(a);
     Writeln('incrementer a=9 de 1 alors a = ',a);
   inc(b,3);
     Writeln('incrementer b=9 de 3 alors b = ',b);
   dec(c);
     Writeln('decrementer c=9 de 1 alors c = ',c);
   dec(d,4);
     Writeln('decrementer d=9 de 4 alors d = ',d);
   b:=succ(entier);
     Writeln('le successeur de 9 est ',b);
   c:=pred(entier);
     Writeln('le predecesseur de 9 est ',c);
   max_int:=high(entier);
     Writeln('le plus grand nombre integer est ',max_int);
   min_int:=low(entier);
     Writeln('le plus petit nombre integer est ',min_int);
   max_long:=high(long);
     Writeln('le plus grand nombre longint est ',max_long);
   min_long:=low(long);
     Writeln('le plus petit nombre longint est ',min_long);
   nb_octet:=sizeof(entier);
     Writeln('la taille en octets de integer est ',nb_octet);
```

```
    nb_octet:=sizeof(long);
      Writeln('la taille en octets de longint est ',nb_octet);
    nb_octet:=sizeof(reel);
      Writeln('la taille en octets de real est ',nb_octet);
    logique:= odd(5);
      Writeln('logique pour un nombre impair = ',logique);
    logique:=odd(4);
      Writeln('logique pour un nombre pair = ',logique);
    entier:=13;
      if odd(entier) then writeln('le nombre ',entier,' est impair')
                     else writeln('le nombre ',entier,' est pair');
end.
```

Résultat de l'exécution du programme exemple :

```
le code ASCII de "c" est 99
l'ordre de b dans l'ensemble des entiers est 5
la valeur absolue de -5 = 5
la valeur absolue de -5.4 = 5.40
le carre de 4 = 16
le carre de 4.2 = 17.64
la racine carre de 4 = 2.00
la racine carre de 4.2 = 2.05
l'arrondi de 15.9 = 16
l'arrondi de 15.5 = 16
l'arrondi de 15.2 = 15
partie entiere de 18954235.758 = 18954235
partie entiere de 14.531 = 0.53100
partie entiere de 105.92 = 105.00
incrementer a=9 de 1 alors a = 10
incrementer b=9 de 3 alors b = 12
decrementer c=9 de 1 alors c = 8
decrementer d=9 de 4 alors d = 5
le successeur de 9 est 10
le predecesseur de 9 est 8
le plus grand nombre integer est 32767
le plus petit nombre integer est -32768
le plus grand nombre longint est 2147483647
le plus petit nombre longint est -2147483648
la taille en octets de integer est 2
la taille en octets de longint est 4
la taille en octets de real est 8
logique pour un nombre impair = TRUE
logique pour un nombre pair = FALSE
le nombre 13 est impair
```

7. Les opérations d’entrée/sortie

7.1. Entrées (Lecture)

Une instruction d'entrée nous permet dans un programme de donner une valeur quelconque à une variable. Ceci se réalise à travers l'opération de lecture. La syntaxe et la sémantique d'une lecture est comme suit :

Algorithme	**PASCAL**	**Signification**
Lire(id_var)	read(id_var); readln(id_var);	Donner une valeur quelconque à la variable dont l'identifiant *<id_var>*.
Lire(iv1, iv2, …)	read(iv1, iv2, …);	Donner des valeurs aux variables <iv1>,<iv2>, …

Remarque :

Lors de la lecture d'une variable dans un programme PASCAL, le programme se bloque en attendant la saisie d'une valeur via le clavier. Une fois la valeur saisie, on valide par la touche entrée, et le programme reprend l'exécution avec l'instruction suivante.

7.2. Sorties (Écriture)

Une instruction de sortie nous permet dans un programme d'afficher un résultat (données traitées) ou bien un message (chaîne de caractères). Ceci se réalise à travers l'opération d'écriture.

La syntaxe et la sémantique d'une écriture est comme suit :

Algorithme	**PASCAL**	**Signification**
Ecrire(id_var) Ecrire(id_const) Ecrire(valeur) Ecrire(expression) Ecrire(‘message’)	write(id_var); write(id_const); write(valeur); write(expression); write(‘message’);	- Afficher la valeur d'une variable - Afficher la valeur d'une constante - Afficher une valeur immédiate - Afficher la valeur calculée d’une expression - Afficher un message tel qu’il est
	writeln(id_var); writeln(id_const); writeln(valeur); writeln(expression); writeln(‘message’);	Même fonction que write mais parès l’affichage le curseur retourne à la ligne. (pointe au début de la ligne suivante).

Exemple :

Programme Pascal	Résultat de l’exécution
`Program affiche ;` `Const` `Max=50 ;` `Var` `A : integer ;` `X : real ;` `Begin` `A :=10 ;`	

X :=3.14 ;	
Writeln(‘bonjour’) ;	Bonjour
Writeln(2015) ;	2015
Writeln(max) ;	50
Writeln(A) ;	10
Writeln(X) ;	3.1400000000E+00
Writeln(X:0:2) ;	3.14
Writeln(‘la valeur de A = ‘, A) ;	la valeur de A = 10
Writeln(‘la valeur de X = ‘,X:0:2,’ reel’);	la valeur de X = 3.14 reel
End.	

8. Structures de contrôles

En générale, les instructions d'un programme sont exécutées d'une manière séquentielle : la première instruction, ensuite la deuxième, après la troisième et ainsi de suite. Cependant, dans plusieurs cas, on est amené soit à choisir entre deux ou plusieurs chemins d'exécution (un choix entre deux ou plusieurs options), ou bien à répéter l'exécution d'un ensemble d'instructions, pour cela nous avons besoins de structures de contrôle pour contrôler et choisir les chemins d'exécutions ou refaire un traitement plusieurs fois. Les structures de contrôle sont de deux types : Structures de contrôles conditionnelles et structures de contrôle répétitives (itératives).

Structures de contrôle conditionnel

Ces structures sont utilisées pour décider de l'exécution d'un bloc d'instruction : est ce que Ce bloc est exécuté ou non. Ou bien pour choisir entre l'exécution de deux blocs différents. Nous avons deux types de structures conditionnelles :

1.1.1. Test alternatif simple

Un test simple contient un seul bloc d'instructions. Selon une condition (expression logique), on décide est ce que le bloc d'instructions est exécuté ou non. Si la condition est vraie, on exécute le bloc, sinon on ne l’exécute pas. La syntaxe d'un test alternatif simple est comme suit :

si <Condition> **alors** <instruction(s)> **finsi**;	**if** <condition> **then** **begin** <instruction(s)>; **end**;
Exemple : lire(x) **si** x > 2 **alors** x ← x + 3 **finsi** ecrire (x)	read(x); **if** x > 2 **then** **begin** x:= x + 3; **end**; write(x);

1.1.2. Test alternatif double

Un test double contient deux blocs d'instructions : on est amené à décider entre le premier bloc ou le second. Cette décision est réalisée selon une condition (expression logique ou booléenne) qui peut être vraie ou fausse. Si la condition est vraie on exécute le premier bloc, sinon on exécute le second.

La syntaxe d'un test alternatif simple est :

si <Condition> **alors** <instruction(s)1> **sinon** <instruciton(s)2> **Finsi**	**if** <condition> **then** **begin** <instruction(s)1>; **end** **else** **begin** <instruction(s)2>; **end**;
Exemple : lire(x) **si** x > 2 **alors** x ← x + 3 **sinon** x ← x - 2 **finsi** ecrire (x) read(x);	**if** x > 2 **then** **begin** x:= x + 3; **end** **else** **begin** x:= x - 2; **end**; write(x);

1.1.3. Test choix .. dans

Elle évite d'utiliser une trop grande suite de ELSE IF. Cette instruction compare la valeur d'une variable de type scalaire à tout un tas d'autres valeurs constantes.

CHOIX expression **DANS** liste_de_cas1 : instruction1; liste_de_cas2 : instruction2; ..... liste_de_casN : instructionN sinon instruction **finchoix**	**CASE** expression **OF** liste_de_cas1 : instruction1; liste_de_cas2 : instruction2; ..... liste_de_casN : instructionN ; else instruction ; **END ;**

L'instruction i sera exécutée si l'expression appartient à la liste_de_cas i. Les autres ne seront pas exécutées (on passe directement au END). L'expression doit être de type scalaire (pas de réels).

En Turbo Pascal, on accepte une liste_de_cas particulière qui est ELSE (et doit être placée en dernier), pour prévoir le cas où expression n'appartient à aucun des cas cités au dessus.

1.1. Structures de contrôle répétitives

Les structures répétitives nous permettent de répéter un traitement un nombre fini de fois.

Nous avons trois types de structures itératives (boucles) :

a. Boucle Pour (For)

La structure de contrôle répétitive pour (for en langage PASCAL) utilise un indice entier qui varie (avec un incrément = 1) d'une valeur initiale jusqu'à une valeur finale. À la fin de chaque itération, l'indice est incrémenté de 1 d'une manière automatique (implicite).

La syntaxe de la boucle pour est comme suit :

pour <indice>←<vi> **à** <vf> **faire** <instruction(s)> **finPour**;	**for** <indice>:=<vi> **to** <vf> **do** **begin** <instruction(s)>; **end**;
<indice> : variable entière <vi> : valeur initiale <vf> : valeur finale	**For** variable := borne_superieure **downto** borne_inferieur **do** {instruction}; **For** variable := borne_superieure **downto** borne_inferieure **do** **begin** ... **end**;

b. Boucle Tant-que (While)

La structure de contrôle répétitive tantque (while en langage PASCAL) utilise une expression logique ou booléenne comme condition d'accès à la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) donc on entre à la boucle, sinon on la quitte.

La syntaxe de la boucle tantque est comme suit :

tant-que <condition> **faire** <instruction(s)> **finTant-que**;	**while** <condition> **do** **begin** <instruction(s)>; **end**;

<condition> : expression logique qui peut être vraie ou fausse.

On exécute le bloc d'instructions tant que la condition est vraie. Une fois la condition est fausse, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après fin Tant que (après end).

Toute boucle pour peut être remplacée par une boucle tantque, cependant l'inverse n'est pas toujours possible.

c. Boucle Répéter (Repeat)

La structure de contrôle répétitive répéter (repeat en langage PASCAL) utilise une expression logique ou booléenne comme condition de sortie de la boucle : si la condition est

vérifiée (elle donne un résultat vrai : TRUE) on sort de la boucle, sinon on y accède (on répète l'exécution du bloc).

La syntaxe de la boucle répéter est comme suit :

Repéter <instruction(s)> **jusqu'à** <condition>;	**Repeat** <instruction(s)>; **until** <condition>;

<condition> : expression logique qui peut être vraie ou fausse.

On exécute le bloc d'instructions jusqu'à avoir la condition correcte. Une fois la condition est vérifiée, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après jusqu'à (après until). Dans la boucle repeat on n'utilise pas begin et end pour délimiter le bloc d'instructions (le bloc est déjà délimité par repeat et until).

La différence entre la boucle répéter et la boucle tantque est :

- La condition de répéter et toujours l'inverse de la condition tantque : pour répéter c'est la condition de sortie de la boucle, et pour tantque c'est la condition d'entrer.
- Le teste de la condition est à la fin de la boucle (la fin de l'itération) pour répéter. Par contre, il est au début de l'itération pour la boucle tantque.